Thomas Bennett - trb090020

CS 4395.001 - UTD Spring 2023

Text Classification 2

News Category Dataset

Source: https://www.kaggle.com/datasets/rmisra/news-category-dataset

In [1]:
```
import pandas as pd

df = pd.read_json('./News_Category_Dataset_v3.json', lines=True)
df.head()
```

Out[1]:

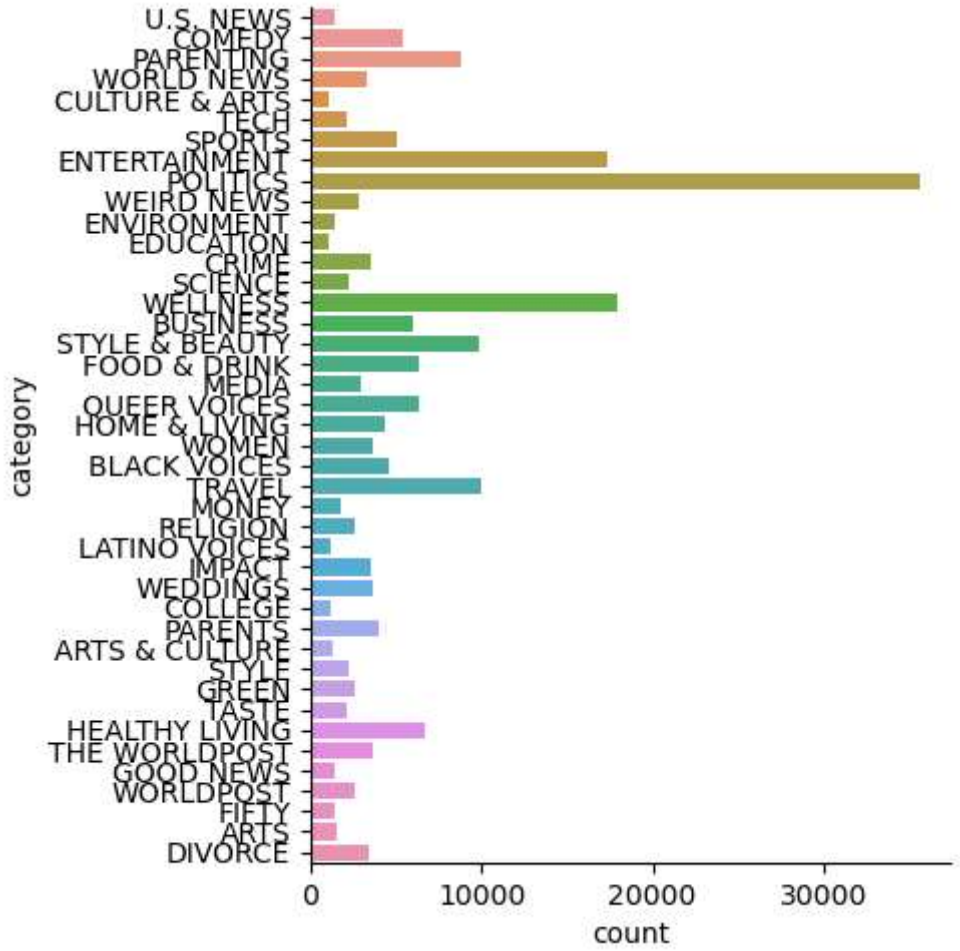
	link	headline	category	short_description	authors	date
0	https://www.huffpost.com/entry/covid-boosters-...	Over 4 Million Americans Roll Up Sleeves For O...	U.S. NEWS	Health experts said it is too early to predict...	Carla K. Johnson, AP	2022-09-23
1	https://www.huffpost.com/entry/american-airlin...	American Airlines Flyer Charged, Banned For Li...	U.S. NEWS	He was subdued by passengers and crew when he ...	Mary Papenfuss	2022-09-23
2	https://www.huffpost.com/entry/funniest-tweets...	23 Of The Funniest Tweets About Cats And Dogs ...	COMEDY	"Until you have a dog you don't understand wha...	Elyse Wanshel	2022-09-23
3	https://www.huffpost.com/entry/funniest-parent...	The Funniest Tweets From Parents This Week (Se...	PARENTING	"Accidentally put grown-up toothpaste on my to...	Caroline Bologna	2022-09-23
4	https://www.huffpost.com/entry/amy-cooper-lose...	Woman Who Called Cops On Black Bird-Watcher Lo...	U.S. NEWS	Amy Cooper accused investment firm Franklin Te...	Nina Golgowski	2022-09-22

Data Exploration

In [2]:
```
import seaborn as sns

sns.catplot(y='category', kind='count', data=df)
```

Out[2]:
```
<seaborn.axisgrid.FacetGrid at 0x1ae919cfa30>
```



In [3]:
```
print(f'Number of articles in the dataset: {len(df)}')
```

```
Number of articles in the dataset: 209527
```

The News Category Dataset consists of approximately 200,000 news articles in 42 categories. I will create a model using keras to predict what category a news article belongs to.

In [4]:
```
import numpy as np
from sklearn import preprocessing

label_encoder = preprocessing.LabelEncoder()
label_encoder.fit(df['category'])
numerical_categories = label_encoder.transform(df['category'])
numerical_categories[:5]
```

Out[4]:
```
array([35, 35,  5, 22, 35])
```

In [5]:
```
num_df = pd.DataFrame(numerical_categories, columns=['target'])
num_df.head()
```

Out[5]:

	target
0	35
1	35
2	5
3	22
4	35

In [6]:
```
df = pd.concat([df, num_df], axis=1)
df.head()
```

Out[6]:

	link	headline	category	short_description	authors	date
0	https://www.huffpost.com/entry/covid-boosters-...	Over 4 Million Americans Roll Up Sleeves For O...	U.S. NEWS	Health experts said it is too early to predict...	Carla K. Johnson, AP	2022-09-23
1	https://www.huffpost.com/entry/american-airlin...	American Airlines Flyer Charged, Banned For Li...	U.S. NEWS	He was subdued by passengers and crew when he ...	Mary Papenfuss	2022-09-23
2	https://www.huffpost.com/entry/funniest-tweets...	23 Of The Funniest Tweets About Cats And Dogs ...	COMEDY	"Until you have a dog you don't understand wha...	Elyse Wanshel	2022-09-23
3	https://www.huffpost.com/entry/funniest-parent...	The Funniest Tweets From Parents This Week (Se...	PARENTING	"Accidentally put grown-up toothpaste on my to...	Caroline Bologna	2022-09-23
4	https://www.huffpost.com/entry/amy-cooper-lose...	Woman Who Called Cops On Black Bird-Watcher Lo...	U.S. NEWS	Amy Cooper accused investment firm Franklin Te...	Nina Golgowski	2022-09-22

Data Preparation

In [7]:
```
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

stopwords = stopwords.words('english')

X = df['short_description']
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10)
vectorizer = TfidfVectorizer(stop_words=stopwords, max_features=10000)

X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

print(f'Train size: {X_train.shape}')
print(f'Test size: {X_test.shape}')
```

```
Train size: (188574, 10000)
Test size: (20953, 10000)
```

In [8]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models

inputs = keras.Input(shape=(10000,))

dense = layers.Dense(16, activation='relu')

x = dense(inputs)
x = layers.Dense(10, activation='relu')(x)

outputs = layers.Dense(42)(x)

model = keras.Model(inputs=inputs, outputs=outputs, name='functional_model')

model.summary()
```

```
Model: "functional_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 10000)]           0

 dense (Dense)               (None, 16)                160016

 dense_1 (Dense)             (None, 10)                170

 dense_2 (Dense)             (None, 42)                462

=================================================================
Total params: 160,648
Trainable params: 160,648
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
print(type(X_train))
print(type(y_train))
```

```
<class 'scipy.sparse._csr.csr_matrix'>
<class 'pandas.core.series.Series'>
```

Dense Model

In [10]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

X_train = X_train.toarray()
y_train = y_train.to_numpy()

history = model.fit(X_train,
                    y_train,
                    epochs=5,
                    batch_size=128)
```

```
Epoch 1/5
1474/1474 [==============================] - 6s 4ms/step - loss: 2.8889 - accuracy:
0.2750
Epoch 2/5
1474/1474 [==============================] - 6s 4ms/step - loss: 2.3777 - accuracy:
0.3993
Epoch 3/5
1474/1474 [==============================] - 6s 4ms/step - loss: 2.2409 - accuracy:
0.4267
Epoch 4/5
1474/1474 [==============================] - 6s 4ms/step - loss: 2.1737 - accuracy:
0.4395
Epoch 5/5
1474/1474 [==============================] - 6s 4ms/step - loss: 2.1282 - accuracy:
0.4478
```

In [11]:
```
print(type(X_test))
print(type(y_test))
```

```
<class 'scipy.sparse._csr.csr_matrix'>
<class 'pandas.core.series.Series'>
```

In [12]:
```
X_test = X_test.toarray()
y_test = y_test.to_numpy()

test_loss, test_acc = model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy: {test_acc}')
```

```
655/655 - 7s - loss: 2.3109 - accuracy: 0.4143 - 7s/epoch - 10ms/step

Test accuracy: 0.41426047682762146
```

Considering that natural chance has 1/42 (~2.4%) probability of success by guessing randomly, this model performs rather well at categorizing news articles.

RNN Model

In [13]:
```
max_features=10000

rnn_model = models.Sequential()
rnn_model.add(layers.Embedding(max_features, 32))
```

```
rnn_model.add(layers.SimpleRNN(32))
rnn_model.add(layers.Dense(42, activation='sigmoid'))
rnn_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 32)          320000

 simple_rnn (SimpleRNN)      (None, 32)                2080

 dense_3 (Dense)             (None, 42)                1386

=================================================================
Total params: 323,466
Trainable params: 323,466
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
rnn_model.compile(optimizer='adam',
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                  metrics=['accuracy'])
```

In [15]:
```
rnn_history = model.fit(X_train,
                        y_train,
                        epochs=10,
                        batch_size=128)
```

```
Epoch 1/10
1474/1474 [==============================] - 7s 5ms/step - loss: 2.0914 - accuracy:
0.4541
Epoch 2/10
1474/1474 [==============================] - 6s 4ms/step - loss: 2.0576 - accuracy:
0.4600
Epoch 3/10
1474/1474 [==============================] - 6s 4ms/step - loss: 2.0254 - accuracy:
0.4657
Epoch 4/10
1474/1474 [==============================] - 6s 4ms/step - loss: 1.9956 - accuracy:
0.4708
Epoch 5/10
1474/1474 [==============================] - 6s 4ms/step - loss: 1.9691 - accuracy:
0.4751
Epoch 6/10
1474/1474 [==============================] - 6s 4ms/step - loss: 1.9462 - accuracy:
0.4789
Epoch 7/10
1474/1474 [==============================] - 6s 4ms/step - loss: 1.9261 - accuracy:
0.4829
Epoch 8/10
1474/1474 [==============================] - 6s 4ms/step - loss: 1.9082 - accuracy:
0.4857
Epoch 9/10
1474/1474 [==============================] - 6s 4ms/step - loss: 1.8919 - accuracy:
0.4890
Epoch 10/10
1474/1474 [==============================] - 6s 4ms/step - loss: 1.8761 - accuracy:
0.4921
```

In [16]:
```
rnn_test_loss, rnn_test_acc = rnn_model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy: {rnn_test_acc}')
```

```
655/655 - 297s - loss: 3.7460 - accuracy: 0.0457 - 297s/epoch - 453ms/step

Test accuracy: 0.045673649758100 51
```

The RNN Model performed better while training the model than during testing. It would appear that the RNN has overfitted the data.

LSTM Model

In [19]:
```
lstm_model = models.Sequential()
lstm_model.add(layers.Embedding(max_features, 32))
lstm_model.add(layers.LSTM(32))
lstm_model.add(layers.Dense(42, activation='sigmoid'))
lstm_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_2 (Embedding)     (None, None, 32)          320000

 lstm_1 (LSTM)               (None, 32)                8320

 dense_5 (Dense)             (None, 42)                1386

=================================================================
Total params: 329,706
Trainable params: 329,706
Non-trainable params: 0
_________________________________________________________________
```

In [20]:
```
lstm_model.compile(optimizer='adam',
                   loss=tf.keras.losses.SparseCategoricalCrossentropy(),
                   metrics=['accuracy'])
```

In [21]:
```
lstm_history = model.fit(X_train,
                         y_train,
                         epochs=10,
                         batch_size=128,
                         validation_split=0.2)
```

```
Epoch 1/10
1179/1179 [==============================] - 24s 20ms/step - loss: 1.8458 - accuracy:
0.4982 - val_loss: 1.9128 - val_accuracy: 0.4851
Epoch 2/10
1179/1179 [==============================] - 6s 5ms/step - loss: 1.8158 - accuracy:
0.5040 - val_loss: 1.9877 - val_accuracy: 0.4688
Epoch 3/10
1179/1179 [==============================] - 6s 5ms/step - loss: 1.7923 - accuracy:
0.5096 - val_loss: 2.0501 - val_accuracy: 0.4573
Epoch 4/10
1179/1179 [==============================] - 5s 5ms/step - loss: 1.7713 - accuracy:
0.5135 - val_loss: 2.1011 - val_accuracy: 0.4468
Epoch 5/10
1179/1179 [==============================] - 6s 5ms/step - loss: 1.7528 - accuracy:
0.5170 - val_loss: 2.1457 - val_accuracy: 0.4410
Epoch 6/10
1179/1179 [==============================] - 6s 5ms/step - loss: 1.7355 - accuracy:
0.5224 - val_loss: 2.1897 - val_accuracy: 0.4335
Epoch 7/10
1179/1179 [==============================] - 5s 5ms/step - loss: 1.7200 - accuracy:
0.5243 - val_loss: 2.2280 - val_accuracy: 0.4280
Epoch 8/10
1179/1179 [==============================] - 6s 5ms/step - loss: 1.7060 - accuracy:
0.5279 - val_loss: 2.2625 - val_accuracy: 0.4252
Epoch 9/10
1179/1179 [==============================] - 5s 5ms/step - loss: 1.6926 - accuracy:
0.5307 - val_loss: 2.2982 - val_accuracy: 0.4215
Epoch 10/10
1179/1179 [==============================] - 6s 5ms/step - loss: 1.6805 - accuracy:
0.5329 - val_loss: 2.3267 - val_accuracy: 0.4198
```

In [22]:

```
lstm_test_loss, lstm_test_acc = lstm_model.evaluate(X_test, y_test, verbose=2)
print(f'\nTest accuracy: {lstm_test_acc}')
```

```
655/655 - 540s - loss: 3.7359 - accuracy: 0.0292 - 540s/epoch - 825ms/step

Test accuracy: 0.029160501435399055
```

Conclusion

Test accuracies:

Dense Model: 41.4% RNN Model: 45.7% LSTM Model: 2.9%

Clearly the LSTM model has overfitted the data very badly. It can be seen in the epochs of the LSTM training that as the accuracy of the training data improved, the accuracy of the validation data declined.